Day#14 Project #3 Guidance

Undergraduates need to complete (1) a web service endpoint (server), and (2) use consume_REST_API.js to display user data in JSON format.

Graduate students need to do the above first. Then, you need to develop another web application to send a request to the Web Service server from within your userController, and then returns to a user (without going through a view file) in JSON format.

I am providing more guidance below.

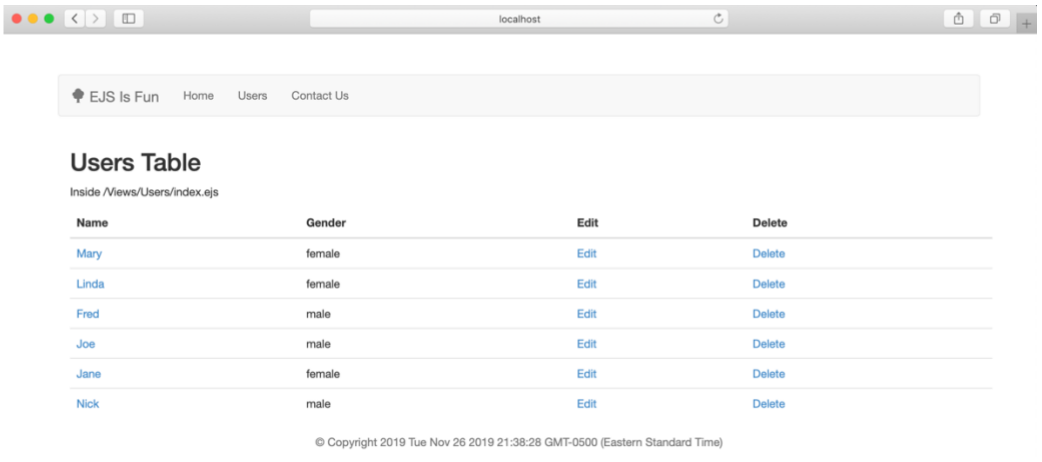
Project #3 Illustration: Endpoint and RESTful Web Service API

This project is a continuation of the previous project. In Project#2, we used a database to store the user data and we accessed the database directly via Mongoose Schema and a Model. In this project, you will still allow that. In addition, you will add a RESTful API to access the database. You will change the route options to allow the access of the user data via the namespace "/api/users".

Part I Change the code to use Mongoose Scheme and a Model to access the database

In the first part of the project, you will begin with accessing the MongoDB database with a Mongoose Scheme and a Model with the name "User" (mapping to a collection "users" by mongoose).

Then, you need to change your project#2 to become an Endpoint of a RESTful API. You only need to implement the route http://localhost:3000/users with the read options. You can use Robo3T to enter some user data offline into your database if your CRUD functions do not work properly. You still need to store some user data into the database.



Users Table

Inside /Views/Users/index.ejs

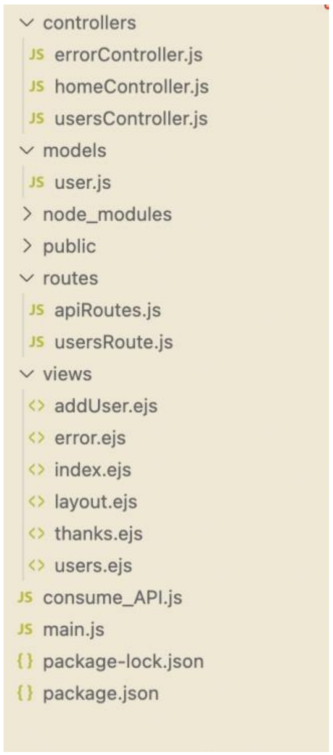
Name	Gender	Edit	Delete
Mary	female	Edit	Delete
Linda	female	Edit	Delete
Fred	male	Edit	Delete
Joe	male	Edit	Delete
Jane	female	Edit	Delete
Nick	male	Edit	Delete

© Copyright 2019 Tue Nov 26 2019 21:38:28 GMT-0500 (Eastern Standard Time)

You do not need to show that the above screen can be displayed with the URL http://localhost:3000/users. We will focus on the link http://localhost:3000/api/users. When this link is entered, the user data will be returned with the JSON format after your Web Service Server is developed successfully.

The procedure for developing the web service server is described as follows. The major steps include (1) main.js must be changed to provide two namespaces, i.e., “/” and “/api”, (2) The route files will be defined in two route files, i.e., apiRoutes.js and userRoutes.js, (3) the homeController and the later a new userController must be provided.

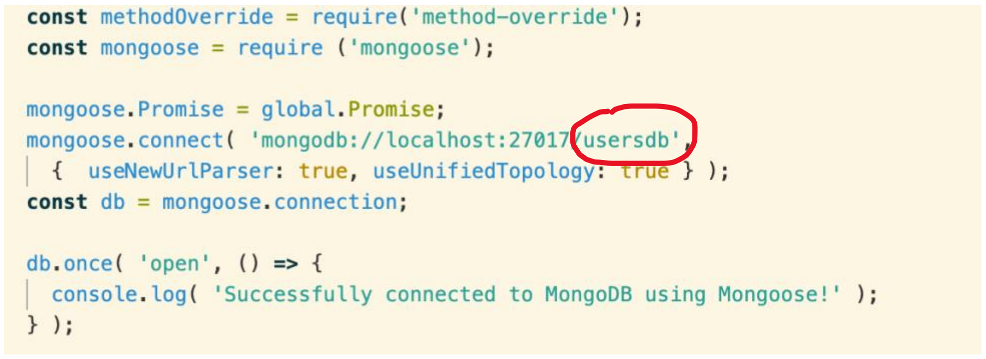
The project structure is going to look like the following:



```
controllers
  errorController.js
  homeController.js
  usersController.js
models
  user.js
node_modules
public
routes
  apiRoutes.js
  usersRoute.js
views
  addUser.ejs
  error.ejs
  index.ejs
  layout.ejs
  thanks.ejs
  users.ejs
consume_API.js
main.js
package-lock.json
package.json
```

Web Server Server (EndPoint)

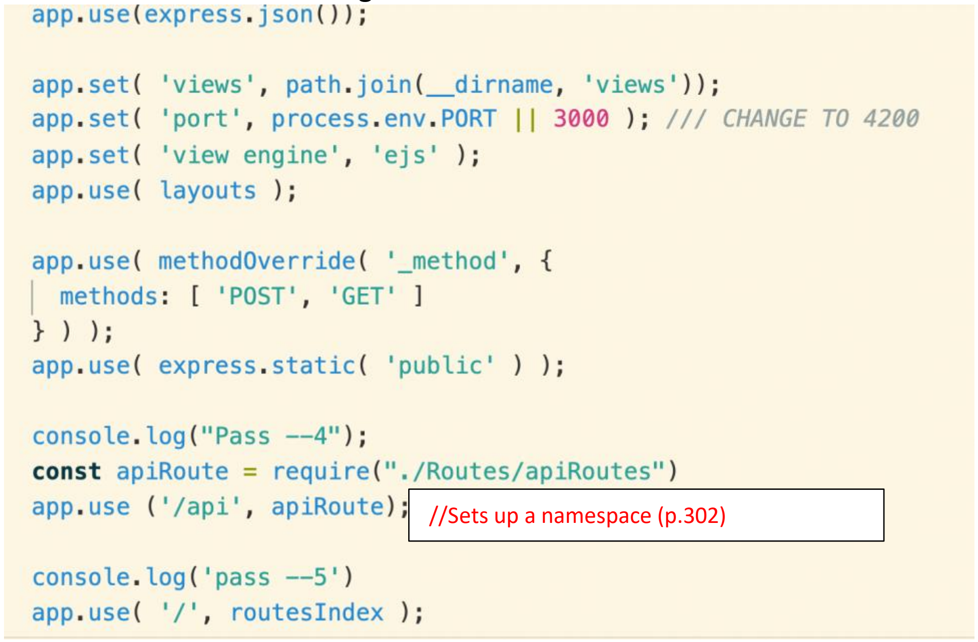
1. In your main.js, you need to add the connection to a mongoose database with the name usersdb, and the Model name “User”. Please use the generic port 27017 so that I can run your code without modifying your program. This is to support the route when a user enters http://localhost:3000/. Note that the “/” symbol after “3000” specifies a route ‘/’.



```
const methodOverride = require('method-override');
const mongoose = require ('mongoose');

mongoose.Promise = global.Promise;
mongoose.connect( 'mongodb://localhost:27017/usersdb',
  {  useNewUrlParser: true, useUnifiedTopology: true } );
const db = mongoose.connection;

db.once( 'open', () => {
  console.log( 'Successfully connected to MongoDB using Mongoose!' );
} );
```

Note: The option useCreateIndex is no longer supported as of January 2022. The database name is "usersdb". We still need to provide a Model with the name "User" (and mapped to the collection "users").

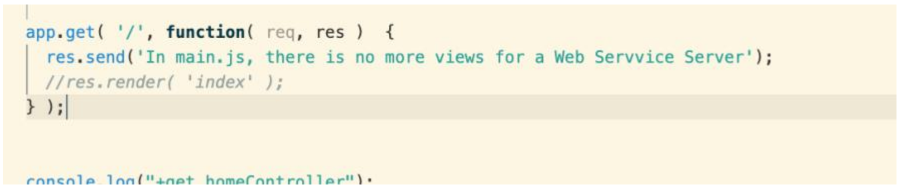
2. Modify main.js to remove those we don't need in Lesson 26, e.g., passport. Your main.js should contain the following:



```
app.use(express.json());

app.set( 'views', path.join(__dirname, 'views'));
app.set( 'port', process.env.PORT || 3000 ); /// CHANGE TO 4200
app.set( 'view engine', 'ejs' );
app.use( layouts );

app.use( methodOverride( '_method', {
  methods: [ 'POST', 'GET' ]
} ) );
app.use( express.static( 'public' ) );

console.log("Pass --4");
const apiRoute = require("./Routes/apiRoutes")
app.use ('/api', apiRoute); //Sets up a namespace (p.302)

console.log('pass --5')
app.use( '/', routesIndex );
```

Then, we need to check what's in the apiRoute.js file. Since we have set up the namespace "/api", we don't need to add the route for "/api/users", we only need to define the route for "/users" inside apiRoute.js. That is the purpose of adding a namespace—saving some typing.

Also, we comment out the following code so that when a user enters http://localhost:3000/, the web browser only displays a string.

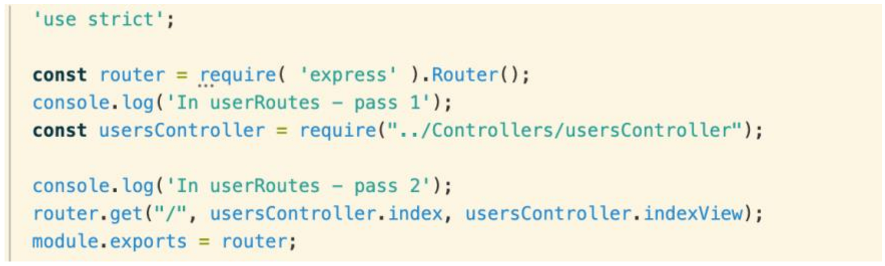


```
app.get( '/', function( req, res )  {
  res.send('In main.js, there is no more views for a Web Servvice Server');
  //res.render( 'index' );
} );


console.log("+get homeController");
```

2. The routesIndex.js and apiRoute.js must be defines and are required in main.js.

```
const apiRoute = require("./routes/apiRoutes");
const usersRoute= require("./routes/usersRoute");
```

3. What do we have in the file “usersRoute.js”?



```
'use strict';

const router = require( 'express' ).Router();
console.log('In userRoutes - pass 1');
const usersController = require("../Controllers/usersController");

console.log('In userRoutes - pass 2');
router.get("/", usersController.index, usersController.indexView);
module.exports = router;
```

Note: The name space “/” has been declared in main.js with app.use. In this file, we need to specify the middleware index() and the indexView() in the userController.js

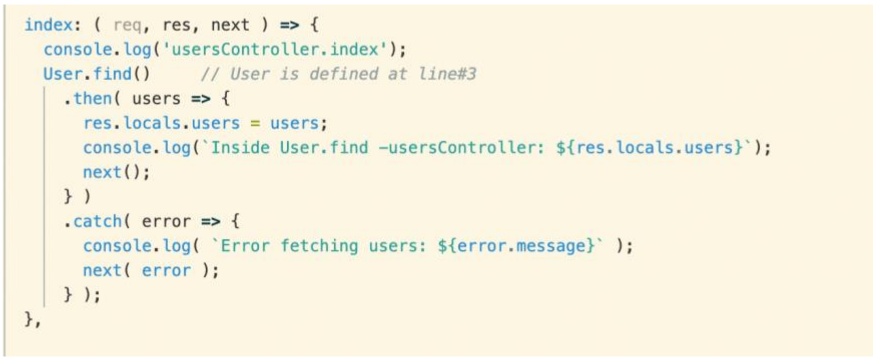
4. What do we have in apiRoute?
We need to set up the “event handling callback function that includes the statement to call res.json(res.locals.users).

```
const router = require("express").Router();
const usersController = require("../Controllers/usersController");

router.get("/users", usersController.index, usersController.respondJSON);
```

Of course, you need to add `module.exports = router;` at the end.

5. Then , you need to change your respondJSON() in usersController to send a request for user data in JSON format. This is explained in Listing 26.3.

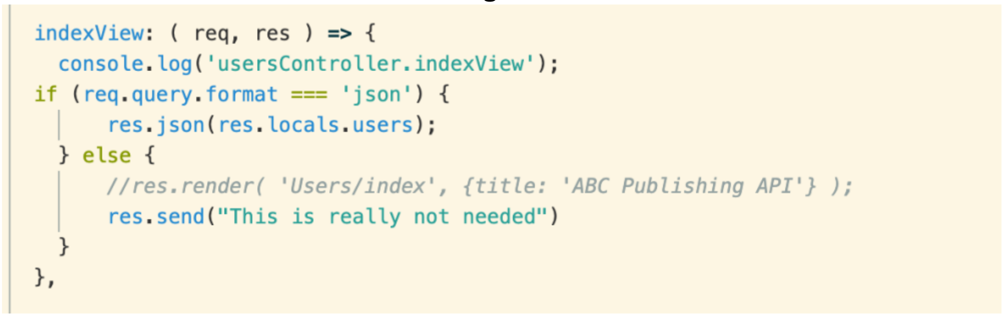
The index() in the usersController looks like the following:



```
index: ( req, res, next ) => {
  console.log('usersController.index');
  User.find()     // User is defined at line#3
    .then( users => {
      res.locals.users = users;
      console.log(`Inside User.find -usersController: ${res.locals.users}`);
      next();
    } )
    .catch( error => {
      console.log( `Error fetching users: ${error.message}` );
      next( error );
    } );
},
```

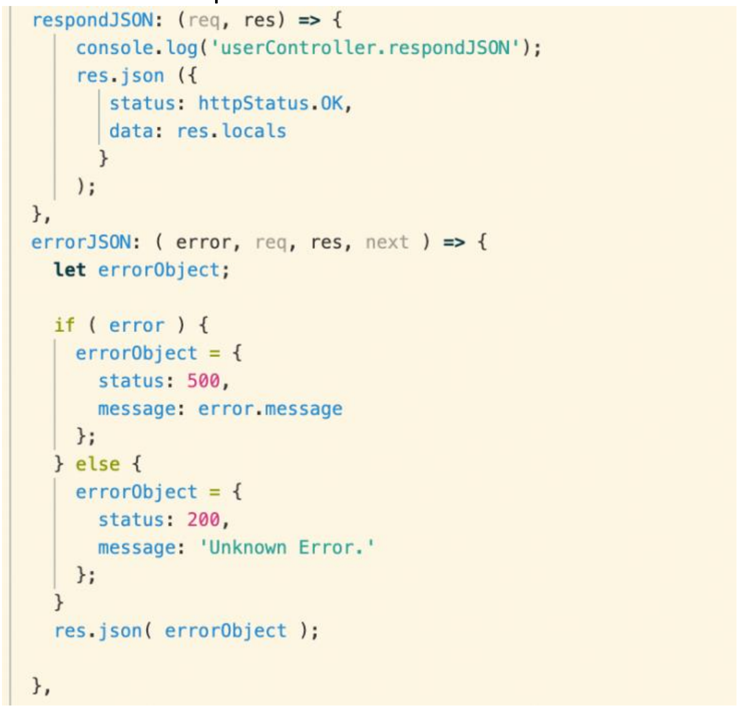
Note: The comma "," after the ending curly brace is needed to enclose these functions in the curly braces as module.exports = { index: …., indexView: …, respondJSON: … }.

The indexView looks like the following:



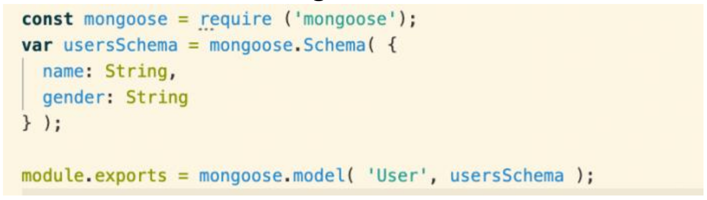
```
indexView: ( req, res ) => {
  console.log('usersController.indexView');
if (req.query.format === 'json') {
      res.json(res.locals.users);
  } else {
      //res.render( 'Users/index', {title: 'ABC Publishing API'} );
      res.send("This is really not needed")
  }
},
```

The function respondJSON insider the usersController looks like the following:



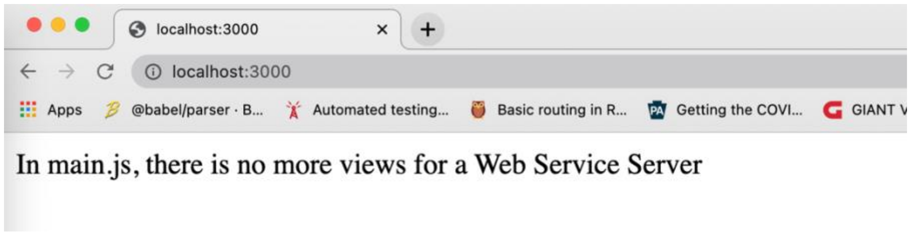
```
respondJSON: (req, res) => {
    console.log('userController.respondJSON');
    res.json ({
       status: httpStatus.OK,
       data: res.locals
      }
    );
},
errorJSON: ( error, req, res, next ) => {
  let errorObject;

  if ( error ) {
    errorObject = {
      status: 500,
      message: error.message
    };
  } else {
    errorObject = {
      status: 200,
      message: 'Unknown Error.'
    };
  }
  res.json( errorObject );

},
```

6. The model “User” must be defined in the file *users.js* and stored under the *Models* folder. It contains the following:



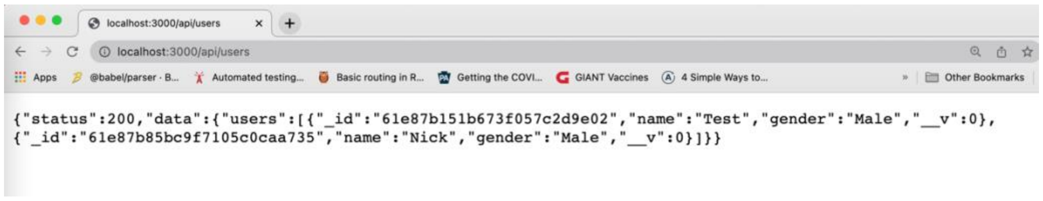
```
const mongoose = require ('mongoose');
var usersSchema = mongoose.Schema( {
  name: String,
  gender: String
} );

module.exports = mongoose.model( 'User', usersSchema );
```

The web service server is ready. You can test it with the consume_REST_API.js. If you bring up a web browser and enter http://localhost:3000/, you will see a string.



In main.js, there is no more views for a Web Service Server

You can test the web service server using a browser by entering http://localhost:3000/api/users.



```
{"status":200,"data":{"users":[{"_id":"61e87b151b673f057c2d9e02","name":"Test","gender":"Male","__v":0},
{"_id":"61e87b85bc9f7105c0caa735","name":"Nick","gender":"Male","__v":0}]}}
```

Part II Access MongoDB Database via the API from an API Client – a standalone program

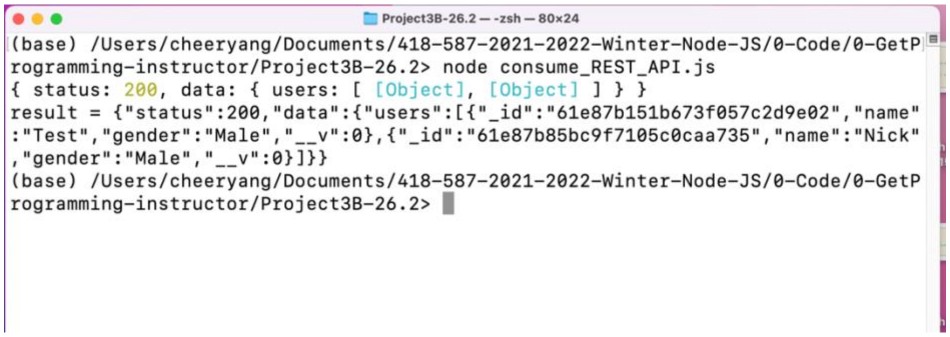
You are going to use a simplified JavaScript code to consume the API. Once the Endpoint is started successfully, you can access the database with the following JavaScript code using the command “node consume_API.js”.

```
//======== consume_API.js =====

var Request = require("request");


Request.get("http://localhost:3000/api/users?format=json", (error, response, body) => {
   if(error) {
       return console.dir(error);
   }
```

```
    console.dir(JSON.parse(body));
    console.log(`result = ${body}`);
});
```

//======= End of Code =========

You need to run this command "node consume_API.js" from a Command Prompt window (Windows) or a Terminal window (MAC OS X).



```
(base) /Users/cheeryang/Documents/418-587-2021-2022-Winter-Node-JS/0-Code/0-GetP
rogramming-instructor/Project3B-26.2> node consume_REST_API.js
{ status: 200, data: { users: [ [Object], [Object] ] } }
result = {"status":200,"data":{"users":[{"_id":"61e87b151b673f057c2d9e02","name"
:"Test","gender":"Male","__v":0},{"_id":"61e87b85bc9f7105c0caa735","name":"Nick"
,"gender":"Male","__v":0}]}}
(base) /Users/cheeryang/Documents/418-587-2021-2022-Winter-Node-JS/0-Code/0-GetP
rogramming-instructor/Project3B-26.2>
```

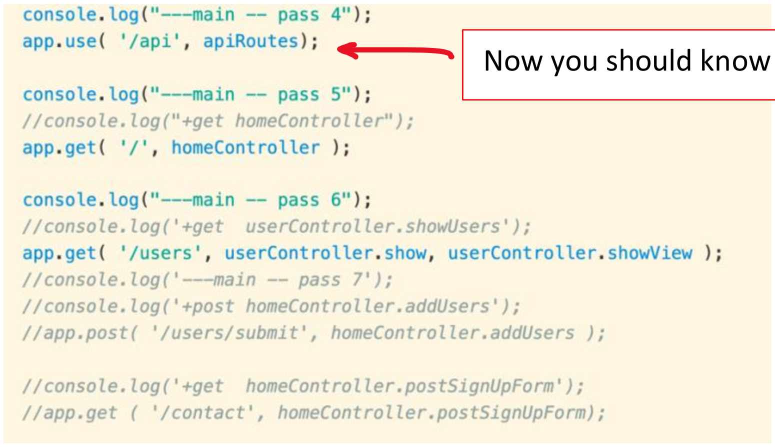
(Undergraduate students stop here.)

Part III Access MongoDB Database via the API from an API Client – an Application

You will now change your code to consume the API from another application code, known as a "client code" here, via port 4200. Once developed, your client code cannot access the database directly. But it will display the user list via the route namespace "/api/users". It is fine to display the user list in JSON format.

This is another Web Application. You will make a copy of the previous web server and modify it. You cannot have the database accessing inside this web application, and you will need to add a request in your homeController, send the request to the web service server inside the indexView(), and display the data in JSON format.

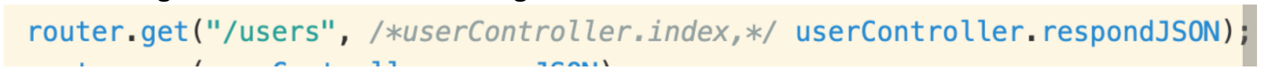
1. Copy the complete project for the web service server to another web application. You need to change the name of the project to *api_client* and call the previous web app *api_server*.
2. Change the homeController (or userController if that's what you call it) to remove all database related code. Change main.js and remove all database related code.
3. Change the main.js and comment out all view handling routes. (You will find that you comment out more code than you write!!)



```
console.log("---main -- pass 4");
app.use( '/api', apiRoutes);

console.log("---main -- pass 5");
//console.log("+get homeController");
app.get( '/', homeController );

console.log("---main -- pass 6");
//console.log('+get  userController.showUsers');
app.get( '/users', userController.show, userController.showView );
//console.log('---main -- pass 7');
//console.log('+post homeController.addUsers');
//app.post( '/users/submit', homeController.addUsers );

//console.log('+get  homeController.postSignUpForm');
//app.get ( '/contact', homeController.postSignUpForm);
```

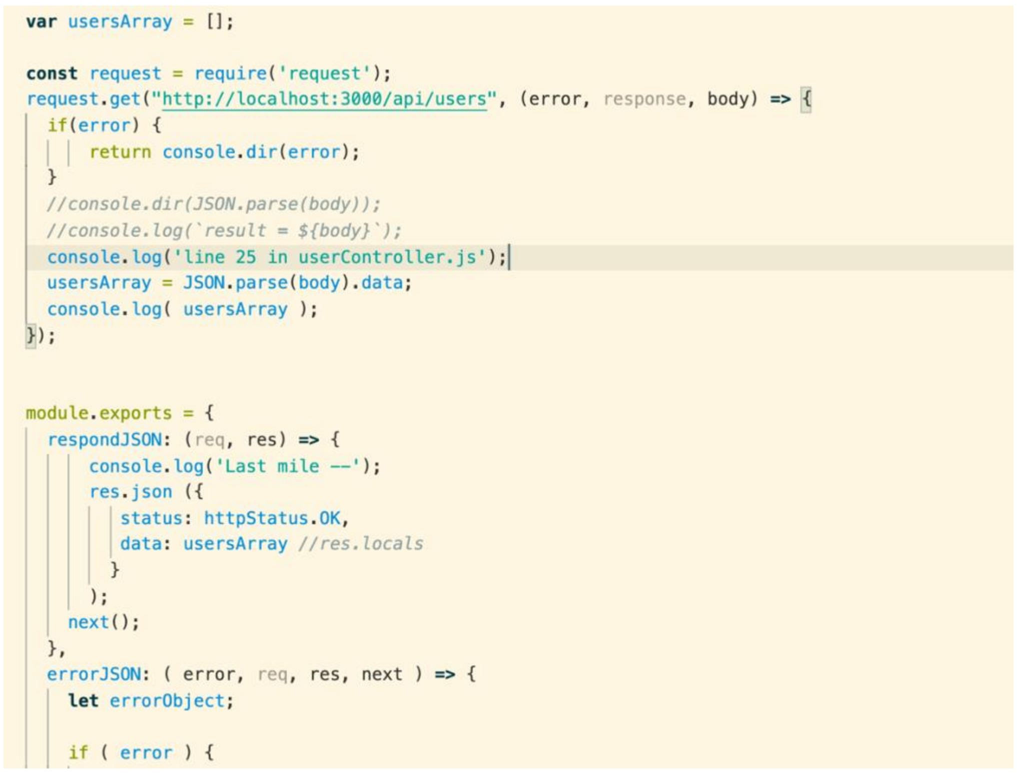
Now you should know what this is for.

4. What do we have in apiRoute.js?
   Check Listing 26.2 and also the following:



```
router.get("/users", /*userController.index,*/ userController.respondJSON);
```

5. You now need to go to userController.js and remove the index() or comment it out. You need to add the respondJSON in the userController.js. I am not giving you the complete code. It is only the code segment. If you copy/paste and try to run it, it will not work; there will be syntax error. You can refer to Listings 27.1 and 27.2.



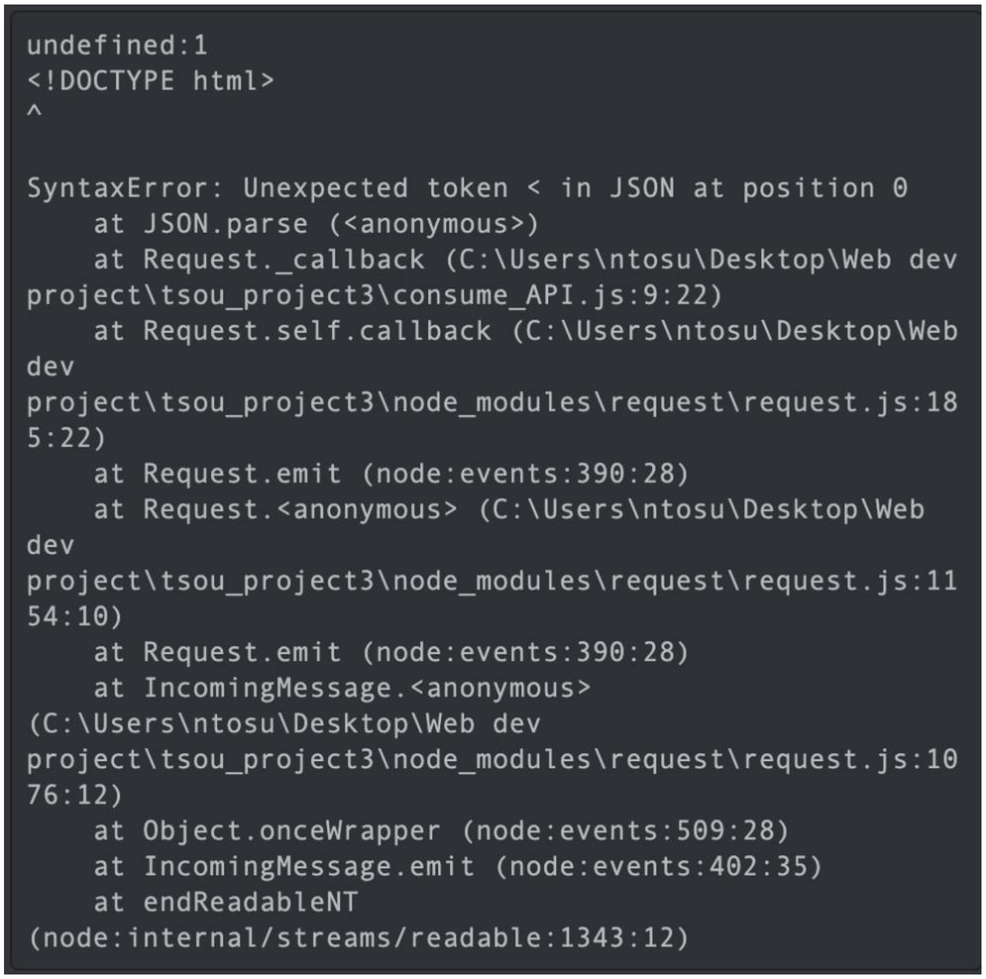
```
var usersArray = [];

const request = require('request');
request.get("http://localhost:3000/api/users", (error, response, body) => {
  if(error) {
    return console.dir(error);
  }
  //console.dir(JSON.parse(body));
  //console.log(`result = ${body}`);
  console.log('line 25 in userController.js');
  usersArray = JSON.parse(body).data;
  console.log( usersArray );
});


module.exports = {
  respondJSON: (req, res) => {
    console.log('Last mile --');
    res.json ({
      status: httpStatus.OK,
      data: usersArray //res.locals
    }
    );
    next();
  },
  errorJSON: ( error, req, res, next ) => {
    let errorObject;

    if ( error ) {
```

Now, check the project description on how to test your web applications. You need to bring up the endpoint #1, the web service server (listening at port 3000), then the endpoint #2 web service client (listening at port 4200). You may need to bring up another Terminal for starting the second endpoint. Then you can bring up a web browser and enter http://localhost:4200/api/users to send the request to the endpoint #1 via the endpoint #2. If you try to enter http://localhost:4200/users, it will not work.

DEBUGGING Q & A

1. What if I received the error of "Module not found --- request"?
   You only need to enter "npm install request" and run "node main.js" again.

2. What is the error like below when I ran "node consume_API.js?"



```
undefined:1
<!DOCTYPE html>
^

SyntaxError: Unexpected token < in JSON at position 0
    at JSON.parse (<anonymous>)
    at Request._callback (C:\Users\ntosu\Desktop\Web dev
project\tsou_project3\consume_API.js:9:22)
    at Request.self.callback (C:\Users\ntosu\Desktop\Web
dev
project\tsou_project3\node_modules\request\request.js:18
5:22)
    at Request.emit (node:events:390:28)
    at Request.<anonymous> (C:\Users\ntosu\Desktop\Web
dev
project\tsou_project3\node_modules\request\request.js:11
54:10)
    at Request.emit (node:events:390:28)
    at IncomingMessage.<anonymous>
(C:\Users\ntosu\Desktop\Web dev
project\tsou_project3\node_modules\request\request.js:10
76:12)
    at Object.onceWrapper (node:events:509:28)
    at IncomingMessage.emit (node:events:402:35)
    at endReadableNT
(node:internal/streams/readable:1343:12)
```

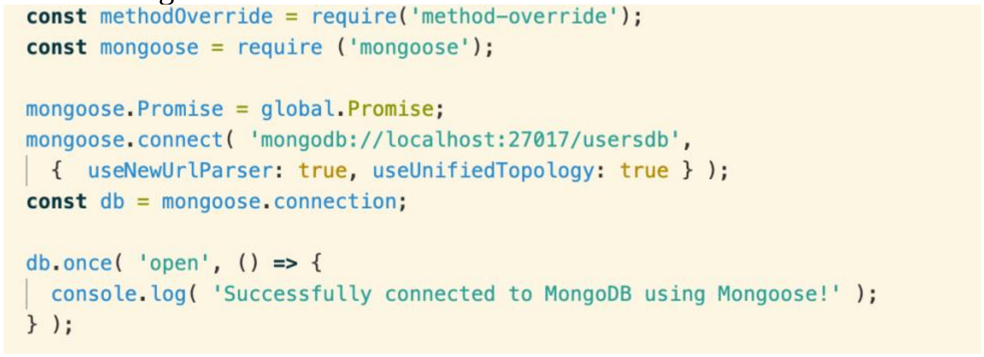
As a general rule of thumb, you need to find out where your web app chocked. You need to add "console.log" in places. In main.js, you can add *console.log("—main.js: pass n");* Here the symbol *n* should be changed to sequential numbers. This is not enough. You may need similar debugging statements in model file, homeController file, etc. You can then run your code to find out where the web app generated the message.

It turns out that you may need to read Lesson 26 in out textbook to review the setup of a namespace, the routing files, and the controller files.

a. Check main.js: For the server, you need to add the database accessing at the beginning, then the route files using namespaces.

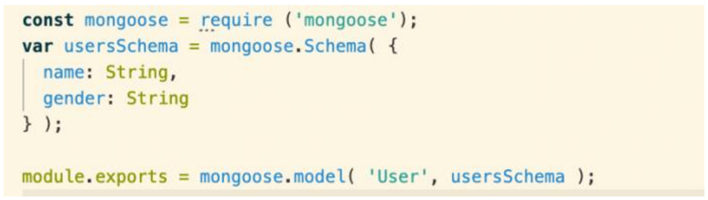
b. Check the model file to see if your model and schema are defined correctly.
c. The most important part is the homeController file. You need to check the functions defined for each route in the route file, e.g., index, indexView, responsJSON, etc.

3. What can I do if I receive the error message "Error: connect ECONNREFUSED ::1:3000eek'?
Your code cannot connect to the database you defined. There are many possibilities. You might not have started the MongoDB database using "mongod –dbpath="./db". I am assuming that the folder at your root is called "db". You might have done it already, but your code to connect the database might not be correct. You can compare your code with the following:



```
const methodOverride = require('method-override');
const mongoose = require ('mongoose');

mongoose.Promise = global.Promise;
mongoose.connect( 'mongodb://localhost:27017/usersdb',
  {  useNewUrlParser: true, useUnifiedTopology: true } );
const db = mongoose.connection;

db.once( 'open', () => {
  console.log( 'Successfully connected to MongoDB using Mongoose!' );
} );
```

You may need to add this at the beginning of your main.js. Then, you may also want to check the model and the schema.



```
const mongoose = require ('mongoose');
var usersSchema = mongoose.Schema( {
  name: String,
  gender: String
} );

module.exports = mongoose.model( 'User', usersSchema );
```

When you run "node consume_API.js", the homeController.js is not executed because the consume_API.js does not sset up any "event handling". What you need to do is to add the mongoose connection code mentioned above to the beginning of main.js. Be sure to add "localhost:27017/api/users". It is omportant to have 27017 as the port.

4. What if I received this error running "node consume_API.js"?
error: => {
^^

SyntaxError: Unexpected token '=>'

Again, find out where the app chocked first and then figure out what the issue was.

5. What if I see the message "${res.locals.users}" being displayed?
The 'tick' quotes need to be replaced by the tick (`) symbol in the console.log().
`console.log(`${res.locals.users}`);`

6. What if I received the error "ERROR occurred: ReferenceError: httpStatus is not defined"?

Add “const httpStatus = require(“http-status-codes”); at the beginning of homeController.js and the enter “npm install require” before executing the ‘node main.js”.

7. What if I received an error message as follows:

```
Error: Route.get() requires a callback function but got a [object
Undefined] at Route.<computed> [as get]
(C:\Users\Hunter\Desktop\WebDev\Project_3\muss_project3\node_modules\expre
ss\lib\router\route.js:202:15) at Function.proto.<computed> [as get]
(C:\Users\Hunter\Desktop\WebDev\Project_3\muss_project3\node_modules\expre
ss\lib\router\index.js:516:19) at Object.<anonymous>
(C:\Users\Hunter\Desktop\WebDev\Project_3\muss_project3\routes\apiRoutes.j
s:6:8) at Module._compile (node:internal/modules/cjs/loader:1101:14) at
Object.Module._extensions..js (node:internal/modules/cjs/loader:1153:10)
at Module.load (node:internal/modules/cjs/loader:981:32) at
Function.Module._load (node:internal/modules/cjs/loader:822:12) at
Module.require (node:internal/modules/cjs/loader:1005:19) at require
(node:internal/modules/cjs/helpers:102:18) at Object.<anonymous>
(C:\Users\Hunter\Desktop\WebDev\Project_3\muss_project3\main.js:6:15)
```

It is complaining about the statement:

Route.get(“/users”, homeController.index, homeController.respondJSON);

Inside the function of get(), it expects multiple argument of which the first one is part of the route “/api/users”. You only need “/users” here because the namespace in this file is “/api”. After that, you need to specify the list of callback functions. But, if the names are not callback functions, the error shows up.
Check if “homeController” is spelled correctly, and if the callback functions “index” and “respondJSON” exist. They must all be callable using *exports* or *module.exports = {…}*